(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 December 2000 (07.12.2000)

PCT

(10) International Publication Number
WO 00/73941 A2

10/825,622

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/US00/14602

(22) International Filing Date: 30 May 2000 (30.05.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/136,764 28 May 1999 (28.05.1999) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, MS PAL01-521, Palo Alto, CA 94303 (US).

(72) Inventors: YALCINALP, Lutfiye, Umit; 1 Debbie Lane, Belmont, CA 94002 (US). KUZNETSOV, Polina; 18361 Vanderbilt Drive, Saratoga, CA 95070 (US).

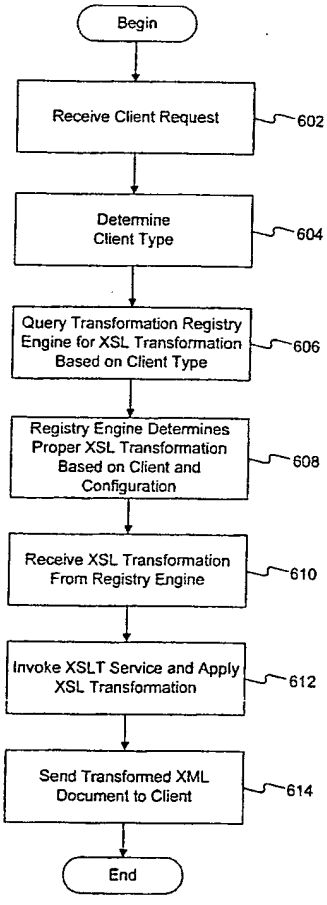
(74) Agents: GARRETT, Arthur, S.; Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P., 1300 I Street, N.W., Washington, DC 20005-3315 et al. (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: TRANSFORMATION REGISTRY SERVICE FOR CONTENT TRANSFORMATION



(57) Abstract: Methods and systems consistent with the present invention solve the inherent problems with existing XSL transformation systems by providing a transformation registry service that serves as a XSL transformation repository. The XSL transformation service enables XSL transformations in applications to deliver XML documents to various clients. Specifically, the transformation registry service maintains mappings for applications, clients, and client configurations. The client configurations are defined based on an application and XSL transformations. The client configurations also allow applications to apply or extend transformations. Each time a client requests a XML document from an application, the application may query the transformation registry service for an appropriate XSL transformation for the client and its configuration. The transformation may then be applied to the XML document and the transformed XML document may be delivered to the requesting client.

WO 00/73941 A2

Published:

— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

TRANSFORMATION REGISTRY SERVICE FOR CONTENT TRANSFORMATION

RELATED APPLICATIONS

Provisional U.S. Patent Application No. 60/136,764 entitled "Transformation Registry Service" filed May 28, 1999, is relied upon and is incorporated by reference in its entirety in this application.

BACKGROUND OF THE INVENTION

A. Field of the Invention

This invention relates generally to data processing systems and, more particularly, to content transformation by using Extensible Style Language (XSL) stylesheets

B. Description of the Related Art

The Internet has been hailed the marketplace of the future. A computer equipped with a communication mechanism such as a modem and telephone connection is nearly all that is necessary to gain access to the Internet and shop for goods and services. A program called a Web browser, such as the NETSCAPE NAVIGATOR from NETSCAPE Corporation, makes it a simple task to traverse the vast network of information available on the Internet and, specifically, its subpart known as the "World Wide Web."

The architecture of the Web follows a client-server model. The terms "client" and "server" refer to a computer's general role as a requester of data (the client) or provider of data (the server). In conventional settings, a Web browser resides in each client and is used to access specially formatted documents that reside on Internet (Web) servers. Web clients and Web servers communicate using a conventional protocol called "HyperText Transfer Protocol" (HTTP).

In operation, a browser opens a connection to a server and initiates a request for a document using a Uniform Resource Locator (URL). The server delivers the requested document, typically in a standard coded format such as the "HyperText Markup Language" (HTML) format. The HTML formatting language incorporates text and graphics into a document by using "tags." HTML tags are codes that identify elements in a document, such as headings or fonts,

for the purpose of formatting information in the HTML document. For example, the tag "<BOLD>" indicates that the text should appear bold face.

In HTML both the tag semantics and the tag set are fixed. An <h1> tag is always a first level heading, while the tag <ati.product.code> is meaningless. The World Wide Web Consortium (W3C) (http://www.wc3.org) has extended the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation, such as stylesheets, to the Web. A stylesheet is a file or form that defines the layout of a document. For example, in the case of Web browsers, the stylesheet may contain instructions that tell the Web browser how to translate the logical structure of a source document into a presentation structure (e.g., display hypertext links in blue, speak emphasized text in a louder voice, or number figures sequentially). However, these changes are rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount. And for people who want to disseminate information widely, features supported by only the latest releases of NETSCAPE NAVIGATOR and INTERNET EXPLORER are not useful.

In response to this limitation the W3C instituted a new formatting language, Extensible Markup Language (XML), that specifies neither semantics nor a tag set. XML is a restricted form of the Standard Generalized Markup Language (SGML) that is more suitable to the Web. SGML is defined by ISO 8879. XML is a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there is no predefined tag set in XML, there cannot be any preconceived semantics. All of the semantics of an XML document are either defined by the applications that process them or by stylesheets.

With XML, developers may create their own customized tags to provide functionality not available with HTML. Further, XML allows content to be separated from its presentation. For example, XML supports links that point to multiple documents, as opposed to HTML links, which can reference just one destination each.

XML documents may be provided to different clients with varied interests and capabilities. For example, a Personal Digital Assistant (PDA) may require

different content and/or presentation of a particular document than that of a Personal Computer (PC) running NETSCAPE NAVIGATOR

The Extensible Style Language (XSL) is one style language used by XML which allowsdifferent clients to receive the same XML documents in different formats. XSL is defined by the WWW Consortium. The Extensible Style Language Transformation (XSLT) language, as part of XSL, allows an XML document to be transformed to another document. The XSL specification language separates style from content when creating XML documents. XSL also allows an XML document to be transformed to another XML document by allowing content transformation. To use XSL with XML documents, a developer creates an XSL stylesheet that describes the transformation of a document written in XSL language, and applies the transformation to multiple XML documents using an XSLT processor. Throughout the specification, XSL transformations are defined as the process that transforms the document by using an XSL stylesheet.

Although an XSL transformation enables developers to transform a particular document to different XML documents, the transformations may be limited if the transformation specific for the application is hard-coded into the application itself. For example, an XML document may need to be transformed and/or styled based on different clients of different applications. Each time an application needs to specify a different type of XSL transformation for a new type of client, the application must be recompiled and/or restarted. Thus, as applications are required to serve new clients and configurations, hardcoding the transformations may be costly within an application that resides on an application server. It is therefore desirable to improve existing XSL transformation systems to provide automatic transformations for new types of clients and configurations unrelated to the application itself.

SUMMARY OF THE INVENTION

Methods and systems consistent with the present invention solve the inherent problems with existing XSL transformation systems by providing a transformation registry service that serves as a XSL transformation repository. The XSL transformation service enables XSL transformations in applications to

deliver XML documents to various clients. Specifically, the transformation registry service maintains mappings for applications, clients, and client configurations. The client configurations are defined based on an application and XSL transformations. The client configurations also allow applications to apply or extend transformations. Each time a client requests a XML document from an application, the application may query the transformation registry service for an appropriate XSL transformation for the client and its configuration. The transformation may then be applied to the XML document and the transformed XML document may be delivered to the requesting client.

Additionally, the transformation registry service provides a facility for developers to enable their applications, to publish and register new XSL transformations, to obtain information for existing transformations, and retrieve XSL stylesheets instead of hard-coding the transformation within the application. The transformation registry service also allows various configurations for clients and applications to be specified or modified.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an implementation of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings.

Figure 1 depicts a data processing system suitable for practicing methods and systems consistent with the principles of the present invention;

Figure 2 depicts a more detailed diagram of the client depicted in Fig. 1;

Figure 3 depicts a more detailed diagram of the application server depicted in Fig. 1;

Figures 4A-4C depict exemplary transformation registries consistent with the principles of the present invention;

Figures 4D and 4E depict exemplary DTDs consistent with the principles of the present invention;

Figure 5 depicts a flow chart of the steps performed by the application server of Fig. 1 when creating a transformation registry in a manner consistent with the principles of the present invention; and

Figure 6 depicts a flow chart of the steps performed by the data processing system of Fig. 1 when providing XML documents to clients in a manner consistent with the present invention.

DETAILED DESCRIPTION

The following detailed description of the invention refers to the accompanying drawings. Although the description includes exemplary implementations, other implementations are possible, and changes may be made to the implementations described without departing from the spirit and scope of the invention. The following detailed description does not limit the invention. Instead, the scope of the invention is defined by the appended claims. Wherever possible, the same reference numbers will be used throughout the drawings and the following description to refer to the same or like parts.

Overview

Systems and methods consistent with the present invention provide a transformation registry service for developers to publish XSL transformations for specific clients and applications, such as applications running on servers. A developer uses an interface associated with the transformation registry service to register applications that require content transformations (of XML documents) on a server, to register clients associated with an application, and to provide mappings between the clients and stylesheets. Each stylesheet describes one or more transformations to apply to an XML document.

Systems and methods consistent with the present invention enable applications that provide XML documents, such as a calendar application, to provide different XML documents based on the type of client. By enabling applications to interact with the transformation registry service, the expensive creation of content tailored for a client may be handled by way of the application and the transformation registry service. Thus, the application may to evolve over time by adding new clients or changing the stylesheets that transform content

independently of the application that generates the content for an XML document.

Each time a client accesses an application that refers to an XML document for an application, a stylesheet is applied to the XML document so that the client may view the XML document tailored for that specific client. For example, a PDA client, with specific memory requirements, may receive an XML document containing limited textual information, and possibly no graphical or audio information. On the other hand, a PC client may receive an XML document containing full content. In another example, a client capable of using the Hypertext Transport Protocol (HTTP) may receive XML documents in an HTTP format.

As explained, the transformation registry service may support multiple applications, clients, and client configuration. Each application, client type, and client configuration is stored as an object in the transformation registry service. Each application may have a set of associated clients namable and callable as needed by the associated application. Each client may have multiple configurations that are supported for that client.

Configurations of a client of an application are specified with different degrees of specificity. For example, all NETSCAPE browser clients may be specified as one client type or the clients may be specified with respect to the platform where the browser client resides in the registry (e.g., Unix, PDA, PC). The developer may also choose the degree of specificity of the client description suitable for the application. To enable this, each configuration is either specified by a name or a set of parameters and values. The parameters of a configuration and their values are specific to a particular application, such as "version" or "ostype" (describing an operating system name). However additional mechanisms can be built on top of this specification to enable common clients and configurations and to provide mappings of common configurations to be described canonically in this architecture. Thus, the architecture allows mapping and searching for a specific type of client with different configurations based on the tree structure of the client specification in the registry.

The transformation registry service also contains default settings for applications, clients and configurations that are unknown to the transformation registry service. The default settings indicate "fallback" configurations for applications and clients of the transformation registry service. The default settings are further described below.

To access the transformation registry service, a client first requests a URL from an application. Based on the client type, the application queries the transformation registry service for a XSL transformation for that application, client type, URL, and client configuration. The client type, URL and client configuration may be found in a protocol request identifier, such as the User-Agent HTTP header. The application may use a client lookup service connected to the transformation registry service to map the received protocol request identifier to a client specification in the registry. Based on the client lookup, the transformation registry service locates a client based on a canonical mapping of the client specification derived from the header. More information on the client lookup service is described below.

The transformation registry service locates the appropriate XSL transformation for the client requesting the URL and returns the XSL transformation to the application. To do this, the application queries the transformation registry service to find a client configuration that applies to the current client requesting an XML document to be transformed.

The transformation registry service provides a number of benefits over traditional XSL transformation systems. The transformation registry service maintains XSL transformations for multiple applications, clients, and configurations. This way, any type of client may request and receive data in a format suitable for that type of client. Each time a client requests an XML document from an application, the transformation registry service provides an appropriate transformation for that client and document based on a best possible configuration that applies to the client.

The transformation registry service provides a facility to represent content in the registry as an XML document, to publish XML documents as the content

of the registry, to receive additional content as XML documents. This may be done through an interface, such as a Web interface.

The transformation registry service also provides a programmable interface for developers to access and submit queries for available XSL transformations. Applications may update the registry with new XSL transformations by accessing the interface. Applications may create a definition for new applications and clients, create multiple client configurations, register stylesheets that specify a transformation for XML documents, and modify or remove the stylesheets from the registry to enable appropriate transformations for an application.

Moreover, the transformation registry service allows different clients corresponding to applications to be configured so that an application can specify new clients overtime. Therefore, applications can dynamically evolve to support new and/or different client, configurations or transformations overtime using the API.

The transformation registry service also allows multiple applications to utilize content transformation by more than one application.

The transformation registry service helper client lookup service provides tailoring for applications which can glue different clients and configuration identifiers (in HTTP this is defined as User-Agent string in the header). Further, this service can be configured to provide different parsing techniques to map different User-Agent headers to known client canonical configurations for an application. Canonical configurations of a client are represented as a set of predefined parameters and values in the registry. This service provides a default mapping based on the commonly known HTTP clients to the internal structure of the transformation registry service which is canonical for all applications.

System Components

Fig. 1 depicts a data processing system 100 suitable for practicing methods and systems consistent with the present invention. Data processing system 100 comprises clients 102, application server 104, and XML documents 106, all connected together. A user uses a client 102, such as a PDA device, a PC computer, or a Unix workstation, to request information from and submit

information to application server 104, such as weather reports, spreadsheet data, or XML documents formatted for client 102.

Application server 104 may host any application (e.g., calendar application, or weather service) that interfaces with clients 102 using XML documents 106. Depending upon the type of client requesting an XML document 106, application server 104 returns the appropriate XML document 106. For example, application server 104 may include the Java Embedded Server (JES), available from Sun Microsystems, Inc. XML documents 106 may be located at various locations in network 110. XML documents 106 may also be stored in application server 104.

Figure 2 depicts a more detailed diagram of client 102, which contains a memory 220, a secondary storage device 230, a central processing unit (CPU) 240, an input device 250, and an optional video display 260. Memory 220 may include a browser 222 that allows users to interact with application server 104 by transmitting and receiving files, such as Web documents. An example of a browser suitable for use with methods and systems consistent with the present invention is the NETSCAPE NAVIGATOR browser, from Netscape Corp. Instead of a browser 222, clients 102 may contain a separate service to enable the client to connect to application server 104, such as the HTTP protocol.

As shown in Figure 3, application server 104 includes a memory 302, a secondary storage device 314, a CPU 318, an input device 320, and an optional video display 322. Memory 302 includes application 304, transformation engine 306, registry 308, XSLT service 310, and client lookup service 312. Application 304 receives client requests and provides XML documents 106 to those clients. Transformation engine 306 maintains mappings for all XSL stylesheets, responds to queries from application 304, publishes new registries 308, and runs an informative servlet that shows the content of registry 308. A mapping refers to a relationship between each of the XSL stylesheets and any other element (e.g., application, client, configuration). A servlet is a program such as a Java servlet that runs on a server. Transformation engine 306 may contain a Web interface, Application Program Interface (API), or other input interface. An API is a set of routines, protocols, or tools for communicating with software

applications. APIs provide efficient access to registry 306 without the need for additional software to interface with the file. The specific components of transformation engine 306 are described below.

Also included in memory 302 is registry 308. Registry 308 contains the various XSL transformations for clients, configurations, and applications. An exemplary data model for registry 308 represented as a tree is depicted in Figure 4A. The tree structure enables registry 308 to be easily manipulated. Each node in the tree provides information about itself, and about its children. Nodes may be added or removed from registry 308. Registry 308 contains multiple applications 402, such as "WeatherService." Each application 402 may contain multiple clients 404, such as Mozilla" (a browser client), "sprinkler" and "thermometer." Each client 404 has multiple configurations 406, such as "osname," "version," or "platform." Each configuration 406 is described by a set of parameters and values. For each configuration 406, a URL 408 describes the source of an XML document (or a set of documents) that may be requested by a client 102, and an XSL stylesheet 410 describes a transformation to apply to an XML document of a particular configuration 406 to obtain the final transformed document. One skilled in the art will appreciate that more than one XSL stylesheet may be used for a single URL.

Memory 302 also contains an Extensible Style Language Transformation (XSLT) service 310. XSLT service 310 is a service that applies an XSL transformation to an XML document given a stylesheet written in XSL. XSLT service 310 applies XSL transformations to XML documents 106.

Finally, memory 302 contains a client lookup service 312 used to map a protocol request identifier to a client specification described in registry 308. That is, lookup service 312 is a helper service that maps an identification string (e.g., HTTP user-agent, or any other client identifier) to a configuration 406 in registry 308 for a specific application 402 and client 404. Lookup service 312 also interacts with application 304 to determine how any headers received from a client (e.g., a HTTP user-agent header from a client browser) should be parsed and mapped to a canonical form in registry 308. The canonical form of client 102 is published in the transformation registry APIs and may be utilized by

application 304 to map an associated client to a canonical form when they are specified for an application 404. The default form of a header string contains parameters, such as client name, version number, OS type, version number, or OS name. The header string is mapped to a client configuration 406 in the registry based on the parameters in the header string. Thus, lookup service 312 may map both known configurations, and unknown configurations. That is, since browser vendors use the User-Agent header string to designate a client and its configuration, client lookup service 312 may map all known variations of configurations to a canonical representation in registry 308 where each configuration is represented by a set of parameters for known configurations. Mapping of a protocol request identifier to a canonical form is essential since different vendors may use different naming conventions to identifying clients and configurations in protocol request identifiers, such as User-Agent in HTTP. To support clients from different vendors, the identifiers must be mapped to a canonical form for searching or matching.

For example, a client that provides a "Mozilla" request, (e.g., a request from a browser, such as Netscape Navigator, Internet Explorer, or HotJava), lookup service 312 may simply use the canonical form to represent a client configuration. A header string may be mapped to a set of configuration paramaters as follows:

type: the browser's type (e.g., "HotJava");
version: the browser's version (e.g., "3.0");
osname: the operating system name (e.g., "Solaris");
osversion: the operating system version (e.g., "2.x");
osarch: the operating system architecture (e.g., "Sparc"); and
lang: the locale or language (e.g., "en").

If a client 102 provides a Mozilla request or an unknown request, the client name and a version string needs to be parsed from the header (e.g., "Sprinkler/1.0"). Lookup service 312 may use the version string and/or client name to locate an acceptable configuration (e.g., a configuration with a client named "Sprinkler" or a version "1.0").

If the version and client name do not exist as an identical matching pair in registry 308, client lookup service 312 may use a best matching algorithm to locate an acceptable configuration 406. That is, based on the canonical mapping of these defined parameters by parsing the parameters, lookup service 312 creates a configuration query to locate a best-matching configuration. The best-matching technique may use predetermined rules to locate an acceptable configuration. For example, among all the matching configurations in registry 308, the best matching configuration may be defined to be the one by the following rules in the order of precedence:

The configurations that have the largest number of matching parameters

The configurations that have the smallest number of non existent parameters in the configuration query.

The default configuration.

An exemplary best matching algorithm is further described in Appendix A

Finally, if no identical or best-matching configuration exists in registry 308, a default configuration may be used by client lookup service 312. The default configuration is further described below with reference to Figure 4C.

Secondary storage device 314 contains static registry 316. Static registry 316 is an XML representation of registry 308. Static registry 316 serves as a backup that may be easily loaded into memory 302. An exemplary representation of static registry 316 parallel to registry 308 is depicted as an XML document in Figure 4B that describes a weather report application having a sprinkler, thermometer, and two different browser clients (a generic Mozilla client and a PDA client running Windows CE).

Figure 4C is an exemplary XML representation of a default application 402, client 404, and configuration 406 that applies a generic stylesheet to an XML document 106. Each entry name is associated with by a default entry ("*"). For example, if a configuration 408 cannot be found in registry 308 after a best match algorithm, application 304 may use the default configuration.

APIs

Methods and systems consistent with the present invention include a set of APIs (interfaces) used by developers to facilitate access to transformation

registry 308. An overview of exemplary APIs for use with methods and systems consistent with the present invention is provided below. A more complete listing is provided in Appendix A. The APIs are described using the Java language and follow the conventions of documenting APIs using the Java language. The appendix also describes the semantics of each method call in each API.

Package com.sun.lhs.service.transformregistry provides a facility to partition and register URLs, devices, configurations, and transformations.

Interface TransformationRegistryService runs as a service and provides capability to publish, modify, add, or delete XML documents that describe registry 308. The TransformationRegistryService utilizes a specific Document Type Definition (DTD) that list various configurations and applications as content in registry 308. A DTD is a type of file associated with XML documents and define how the markup tags within the XML document will be interpreted by an application presenting the document. Each XML representation in registry 308 utilizes a specific DTD. Figures 4D-4E depict exemplary DTDs for use with methods and systems consistent with the present invention.

Interface TransformationRegistryFactory enables a developer to create elements in the registry. This interface is called to create an Application node, Client node, or Configuration node.

Interface Application provides access to various applications part of the registry. This interface is used to create and modify applications 402 and add new clients 404 to an application 402. For example, application 402 may represent an application 304 and be a weather report application.

Interface Client provides access to various clients 404 part of the registry. Clients 404 belong to applications 402. Clients 404 also contain various configurations 406. For example, a weather application represented by application 402 might contain a Netscape browser or a PDA as a client 404.

Interface Configuration defines a set of XSL transformations to be used when a client 102 requests data. As explained, each configuration 406 is defined by a set of parameters and values that uniquely describe a configuration. The Configuration interface allows developers to locate stylesheets 410 for a specific source URL 408 in registry 308, as well as obtain and set parameters of a

specific configuration 406. For example, a Netscape client 404 may contain two configurations 406 (e.g., a 4.x version, and a 5.x version). Thus, the version is provided as a differentiating parameter that designates a specific configuration 406. If application 402 requires a transformation to apply to a default configuration, this case is specifically marked by a "*" in configuration 406.

Interface ConfigurationQuery takes a string and locates for matching configurations in various applications, or clients. This interface may be used to locate an appropriate XSL transformation for a requesting client and application.

Interfaces DuplicateConfigurationException, ElementAttachedException, and RegistryDefinitionException return error codes when various errors occur within the registry.

Publish Process

Figure 5 depicts a flow chart of the steps performed when publishing a registry in accordance with methods and systems consistent with the present invention. Figure 5 depicts three branches of execution (new applications, incremental changes, or new application). In the new applications branch 500, any previous existing content of registry 308 will be replaced with the content of static registry 316. In the incremental changes branch 510, additional registry 308 entries (e.g., applications, clients, or configurations) are added to an existing registry 308. In the new application branch 520, existing registry 308 is not deleted and, instead a new application is added to registry 308.

As shown in new applications branch 500, the publish process is initiated, for example, by creating a static registry 316 as an XML document containing application definition, clients, configurations and stylesheets for multiple applications (step 502). Since the initial form of registry 316 is an XML document (before it is published to registry 308), static registry 316 may be created by a text editor or if desired by any well-known XML authoring tool. XSL stylesheets are also created for multiple applications 304 (step 504). Once static registry 316 and XSL stylesheets for all applications 304 are created, a developer may user the TransformationRegistryService API to load (publish) static registry 316 and the XSL stylesheets into registry 308 (step 506). By loading static registry 316, any content already in registry 308 is deleted. In an alternative

embodiment, registry 308 may contain links (e.g., URLs) to the specified XSL stylesheets. Also in step 506, application(s) 304 may be installed on application server 104. Once application(s) 304 are installed on application server 104, application(s) 304 may begin providing XML documents 106 to requesting clients 102.

Once registry 308 contains the XSL stylesheets, transformation engine 306 may be initiated (step 508). That is, the developer may call the TransformationRegistryService API to publish registry 308, thus enabling applications 304 to query registry 308 each time application 304 receives a request from a client 102. Next, transformation engine 306 enters a "ready" state. In the ready state, content in registry 308 may be changed by publishing a completely new registry in new applications branch 500, by publishing additional applications in application branch 520, or by performing incremental changes to registry in incremental branch 510.

In incremental change branch 510, additional registry 308 entries (e.g., applications 402, clients 404, or configurations 408) may be created and/or modified by a developer by using components of the TransformationRegistryFactory API (step 512). Similar to application branch 500, transformation engine 306 next enters a "ready" state.

As shown in new application branch 520, the publish process is initiated, for example, by creating a static registry 316 as an XML document containing an application definition containing clients, configurations and stylesheets (step 522). An XSL stylesheet may also created for a particular application 304 (step 524).

Once static registry 316 and a XSL stylesheet for an application 304 is created, a developer may user the TransformationRegistryService API to load (publish) static registry 316 and the XSL stylesheet into registry 308 (step 526). Unlike new applications branch 500, by loading static registry 316, any content already in registry 308 is preserved. That is, step 526 simply adds an additional application to registry 308. Also in step 526, application 304 may be installed on application server 104. Once registry 308 contains the XSL stylesheets,

transformation engine 306 may be initiated (step 528). Similar to application branch 500, transformation engine 306 next enters a "ready" state.

In one example consistent with the present invention, a developer prepares an XML representation of the registry and utilizes a servlet that installs an XML document from static registry 316 to registry 308. This may be used when initially loading registry 308. The developer may also change registry 308 or even registry 314 with any well-known Web browser using the servlet. In another example, an application 304 may use an API to publish additional registry entries, such as clients, configurations or associated stylesheets to an already existing registry 308. Since each application 304 is partitioned in registry 308, application 304 may specify new clients and XSL transformations without disturbing existing XSL transformations.

Registry Service

As shown in Figure 6, the transformation registry service is initiated, for example, by application 304 receiving a request from a client 102 for an XML document 106 (step 602). The request includes information pertaining to that client. For example, a HTTP request (e.g., from a NETSCAPE NAVIGATOR or MICROSOFT EXPLORER browser, or a PDA client) includes identification information about that client. Application 304 may then determine a requesting client type or configuration to create a query for transformation engine 306 based on information supplied in the HTTP header (step 604). That is, application 304 may parse the protocol identifier string (e.g., User-Agent) to determine client 404 and configuration 406 that best matches the requesting client's configuration. Alternatively, application 304 may use client lookup service 312 to help identify a correct configuration 406 by using a best matching algorithm or using a default configuration if no match occurs, both of which are described above. Client lookup service 412 may also be used when configured for providing canonical mappings for an application. This service can be configured at the time of the application installation.

Once the client type is determined (or a default configuration or best match is supplied), application 304 may query transformation engine 306 for an XSL transformation based on the client type, configuration, and application (step 606).

Alternatively, application 304 may query transformation engine 306 for a default application 402, client 404, or configuration 406. In that case, a "*" may be put in place of all parameters.

Once application 304 queries transformation engine 306 for an XSL transformation, transformation engine 306 traverses registry 308 for one or more appropriate XSL stylesheets and configurations for the requesting client (step 608). It may be that the requesting client requires multiple XSL stylesheets to display the XML document.

For example, the application may query transformation engine 306 for configurations 406 associated with an application 402, such as "calendar." Engine 306 searches clients 404 by name, such as "Mozilla" in registry. Engine 306 may also create a configuration query based on the client type and application type. This way, engine 306 may locate the correct version and platform of the browser as parameters and their values, and the requested URL designated by the client. The registry will return the best matching URL 408 that corresponds to this client configuration 406 from registry 308.

Transformation engine 306 then supplies application 304 with the appropriate XSL stylesheets(s) (step 610). For example, transformation engine 306 may return a pointer (e.g., URL) to XSL stylesheet 410, or may return the complete stylesheet 410 that correspond to the XSL transformation to apply.

Once received, application 304 may invoke a XSLT service 310 (step 612). XSLT service 310 applies received XSL stylesheets 410 to the requested XML document 106. If more than one stylesheet 410 needs to be applied to the transformation, all stylesheets are applied sequentially where the transformed document as an output of one transformation is used as input to the next stylesheet in sequence. After all transformations, XML document 106 may be sent to a client 102 (step 614).

Conclusion

As described, methods and systems consistent with the present invention provide a transformation registry service that is a transformation repository for multiple XSL stylesheets.

Although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects may be stored on or read from other computer readable media, such as secondary storage devices, like hard disks, floppy disks, and CD-ROM; a carrier wave received from a network like the Internet; or other forms of ROM or RAM. Additionally, although specific components and programs of client computer 102, and various servers have been described, one skilled in the art will appreciate that these may contain additional or different components or programs.

The foregoing description of an implementation of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. In another example, client 102 may be directly connected to application 104.

com.sun.lhs.service.transformregistry

Interface Application

public interface **Application**

An Application is the top-level node of the Transformation Registry hierarchy. An Application has Clients, each of which has Configurations that specify XSL transformations. These transformations are used to change data (requested by a client) into a form appropriate for that client.

Application names must be unique.

A default Application (whose name is "*") may be specified. This default Application will be used by services that are not part of any specific Application. There must be no more than one default Application in the Transformation Registry.

An enumeration of known Applications is available from the TransformationRegistryService.

Method Summary

boolean	addClient(Client client) Add a client to the application.
Client	getClientByName(java.lang.String name) Returns a client.
java.util.Enumeration	getClients() Returns all clients known for this application.
Client	getDefaultClient() Returns the default client.
java.lang.String	getDescription()
java.lang.String	getName()
boolean	isDefaultApplication()
Client	removeClient(java.lang.String clientName) Remove a Client from the Application.

Method Detail

getName

Appendix A

```
public java.lang.String getName()
```

Returns:
The name of the Application

---

getDescription

```
public java.lang.String getDescription()
```

Returns:
The description of the Application (i.e. version info)

---

isDefaultApplication

```
public boolean isDefaultApplication()
```

Returns:
True when the Application is the default Application (the name of this Application is "*")

---

addClient

```
public boolean addClient(Client client)
                  throws ElementAttachedException
```

Add a client to the application. This call fails when there is already a client with the same name in the application.

Note: If the client is already attached to an application and is being added to the same application, this method has no effect and returns true. If the client is already attached to a different application, an ElementAttachedException is thrown.

Parameters:
`client` - The Client to add to the Application
Returns:
true on success, false on failure
Throws:
ElementAttachedException - when the client is already attached to a different Application

---

removeClient

```
public Client removeClient(java.lang.String clientName)
```

Remove a Client from the Application.

Parameters:
`clientName` - The name of the client to remove
Returns:
the client that was removed (null if no such client was in the registry)

Appendix A

getClients

```
public java.util.Enumeration getClients()
```

Returns all clients known for this application.

Returns:
An enumeration of all Clients known to this Application (including default Client, if it exists)

---

getDefaultClient

```
public Client getDefaultClient()
```

Returns the default client. The default client is the client whose name is "*".

Returns:
The default Client, if it exists

---

getClientByName

```
public Client getClientByName(java.lang.String name)
```

Returns a client.

Parameters:
name - The name of the Client
Returns:
The Client with the given name

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Interface Client

public interface **Client**

Clients belong to an Application and specify configurations. Each configuration specifies XSL transformations to be applied when the Client matching that Configuration requests a URI.

Client names must be unique only within their Application.

A default Client (whose name is "*") may be specified. This default Client can be used when the requesting client does not match any known Clients for the Application. There must be no more than one default Client per Application.

An enumeration of known Clients is available from the enclosing Application.

Method Summary

boolean	addConfiguration(Configuration configuration) Add a configuration to the client.
Configuration	getConfigurationByName(java.lang.String name) Find a configuration with the given name.
java.util.Enumeration	getConfigurations() Get an enumeration of the client's configurations.
Client	getCopy(java.lang.String name, java.lang.String description) Create a deep copy of this client.
Configuration	getDefaultConfiguration() Get the client's default configuration.
java.lang.String	getDescription()
java.util.Enumeration	getMatchingConfigurations(ConfigurationQuery query, boolean bestMatchesOnly) Find all matching configurations.
java.lang.String	getName()
boolean	isDefaultClient()
Configuration	removeConfiguration(java.lang.String configurationName) Remove a Configuration from the Client.
java.util.Enumeration	removeConfigurations(ConfigurationQuery query, boolean removeNamedConfigurations) Remove Configurations from the Client.

Method Detail

getName

```
public java.lang.String getName()
```

Returns:
The name of the Client

---

getDescription

```
public java.lang.String getDescription()
```

Returns:
The description of the Client

---

isDefaultClient

```
public boolean isDefaultClient()
```

Returns:
True when the Client is the default Client (the name of this Client is "*")

---

addConfiguration

```
public boolean addConfiguration(Configuration configuration)
                        throws ElementAttachedException
```

Add a configuration to the client.

This call fails when a duplicate configuration already exists. For named configurations, a duplicate configuration is one with the same name. For unnamed configurations, a duplicate configuration is one with the same set of parameter names and parameter values.

Note: If the configuration is already attached to a client and is being added to the same client, this method has no effect and returns true. If the configuration is already attached to a client and it is being added to a different client, an ElementAttachedException is thrown.

Parameters:
`configuration` - The Configuration to add to the Client
Returns:
true on success, false on failure
Throws:
ElementAttachedException - when the configuration is already attached to a different Client

---

removeConfiguration

Appendix A

```
public Configuration removeConfiguration(java.lang.String configurationName)
```

Remove a Configuration from the Client.

Parameters:
`configurationName` - The name of the configuration to remove
Returns:
the configuration that was removed (null if no such configuration existed)

---

removeConfigurations

```
public java.util.Enumeration removeConfigurations(ConfigurationQuery query,
                                                  boolean removeNamedConfigurations)
```

Remove Configurations from the Client. Only configurations whose parameters exactly match those of the query will be removed by this method. If `removeNamedConfigurations` is false, only unnamed configurations are removed.

Parameters:
`query` - A configuration query exactly specifying the parameters of the configurations to be removed.
`removeNamedConfigurations` - whether or not to remove named configurations
Returns:
an enumeration of the removed configurations

---

getConfigurations

```
public java.util.Enumeration getConfigurations()
```

Get an enumeration of the client's configurations.

Returns:
An enumeration of all Configurations for this Client

---

getDefaultConfiguration

```
public Configuration getDefaultConfiguration()
```

Get the client's default configuration.

Returns:
A default Configuration (one whose name is "*"), if it exists

---

getConfigurationByName

```
public Configuration getConfigurationByName(java.lang.String name)
```

Find a configuration with the given name.

Appendix A

Parameters:
name - Configuration name
Returns:
Configuration with matching name

---

getMatchingConfigurations

```
public java.util.Enumeration getMatchingConfigurations(ConfigurationQuery query,
                                                       boolean bestMatchesOnly)
```

Find all matching configurations. This method allows the user to inspect all configurations matching the configuration query.

Parameters:
query - A ConfigurationQuery
bestMatchesOnly - If true, only the "best" matches are returned (see ConfigurationQuery for more information about "best" matches). If false, all matches are returned.
Returns:
An enumeration of Configurations matching the query.
See Also:
ConfigurationQuery

---

getCopy

```
public Client getCopy(java.lang.String name,
                      java.lang.String description)
```

Create a deep copy of this client. The copy can then be added to another Application in the registry.

Parameters:
name - The name of the client copy. If this argument is null, the existing client's name is used.
description - The description of the client copy. If this argument is null, the existing client's description is used.

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Interface Configuration

public interface **Configuration**

A Configuration of a Client defines a set of transcodings (XSL transformations) to be used when Clients request data.

A Configuration may have a name, which must be unique within the enclosing Client. Therefore, a name uniquely specifies the Configuration. A configuration (with or without a name) may also be specified by its parameters.

An enumeration of Configurations is available from the enclosing Client.

Field Summary

static int	ANY Source node applies to either XML or HTML
static int	HTML Source node applies to HTML
static int	XML Source node applies to XML

Appendix A

Method Summary	
Configuration	getCopy(java.lang.String name, java.lang.String description) Create a deep copy of this configuration.
java.lang.String	getDescription()
java.lang.String	getName()
java.lang.String	getParam(java.lang.String paramName) Return the value of the given parameter.
java.util.Enumeration	getParamNames()
java.util.Enumeration	getTransformations(java.lang.String sourceURI, int type) Find the XSL transformations to be applied on the document at the given source URI.
boolean	isDefaultConfiguration()
java.lang.String	removeParam(java.lang.String paramName) Remove the parameter.
void	removeTransformationsForSource(java.lang.String sourceURI, int type) Remove transformations for source URI.
java.lang.String	setParam(java.lang.String paramName, java.lang.String paramValue) Set a parameter value.
void	setTransformationsForSource(java.lang.String sourceURI, int type, java.util.Enumeration transforms) Set the transformations for a source URI.

Field Detail

XML

```
public static final int XML
```

Source node applies to XML

HTML

```
public static final int HTML
```

Source node applies to HTML

ANY

```
public static final int ANY
```

Appendix A

Source node applies to either XML or HTML

Method Detail

getName

```
public java.lang.String getName()
```

Returns:
The name of the Configuration

---

getDescription

```
public java.lang.String getDescription()
```

Returns:
The description of the Configuration

---

getParamNames

```
public java.util.Enumeration getParamNames()
```

Returns:
An enumeration of parameter names

---

getCopy

```
public Configuration getCopy(java.lang.String name,
                             java.lang.String description)
```

Create a deep copy of this configuration. The copy can then be added to another Client in the registry, or modified and added to the same client.

Parameters:
`name` - The name of the client copy. If this argument is null, the existing client's name is used.
`description` - The description of the client copy. If this argument is null, the existing client's description is used.

---

getParam

```
public java.lang.String getParam(java.lang.String paramName)
```

Return the value of the given parameter.

Parameters:
`paramName` - Parameter name
Returns:

Appendix A

Parameter value, or null if no such parameter exists

---

setParam

```
public java.lang.String setParam(java.lang.String paramName,
                                 java.lang.String paramValue)
                          throws DuplicateConfigurationException
```

Set a parameter value.

Note: This call fails (by throwing an exception) when changing this parameter would result in the Configuration being a duplicate of another Configuration in the enclosing Client.

Parameters:
`paramName` - Parameter name (non-null)
`paramValue` - Parameter value (non-null)
Returns:
the previous value of the parameter (or null if no previous value existed)
Throws:
DuplicateConfigurationException - when the change would result in this Configuration being a duplicate in the enclosing Client

---

removeParam

```
public java.lang.String removeParam(java.lang.String paramName)
                                throws DuplicateConfigurationException
```

Remove the parameter.

Parameters:
`paramName` - Name of parameter to remove
Returns:
Parameter value (null if there was no such parameter)
Throws:
DuplicateConfigurationException - when the change would result in this Configuration being a duplicate in the enclosing Client

---

isDefaultConfiguration

```
public boolean isDefaultConfiguration()
```

Returns:
True when the Configuration is the default Configuration (the name of this Configuration is "*")

---

setTransformationsForSource

```
public void setTransformationsForSource(java.lang.String sourceURI,
                                        int type,
                                        java.util.Enumeration transforms)
```

Appendix A

Set the transformations for a source URI. If the source URI already exists, its transformations are replaced with the new ones.

Parameters:

`sourceURI` - The source URI. The hostname in the URI should be fully qualified (with the exception of localhost).
`type` - The type of the source (XML, HTML, ANY).
`transforms` - The list of transformations to apply. Each element should be a String URI.

---

removeTransformationsForSource

```
public void removeTransformationsForSource(java.lang.String sourceURI,
                                           int type)
```

Remove transformations for source URI.

Parameters:

`sourceURI` - The source URI to remove. The hostname in the URI should be fully qualified (with the exception of localhost).
`type` - The type of the source (XML, HTML, ANY).

---

getTransformations

```
public java.util.Enumeration getTransformations(java.lang.String sourceURI,
                                                int type)
```

Find the XSL transformations to be applied on the document at the given source URI.

Parameters:

`source` - The document's URI.
`type` - The type of the source (XML, HTML)

Returns:

An enumeration of XSL transformations (Strings) for the transcoding whose source best matches the given source. The best match is the longest match.

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Interface ConfigurationQuery

public interface **ConfigurationQuery**

A ConfigurationQuery is used to fill in all the known configuration parameters, and to look for matching Configurations in the Client. This is useful when the configuration name is not known (or does not exist).

A configuration query is first created (by using `Application.createConfigurationQuery()`), then made more specific by setting parameters, and finally passed to `Client.getMatchingConfigurations()` or `Client.getTransformationsForConfiguration()`

A ConfigurationQuery matches a Configuration iff, for every parameter defined in the query, either:

- the parameter is not defined in the Configuration
- the parameter has the same value in the Configuration and the ConfigurationQuery

The "best" matching Configuration (of a given Client) is chosen among the matching Configurations according to the following rules (in order of precedence):

1. The Configurations having the largest number of matching parameters.
2. The Configurations having the smallest number of parameters that do not appear in the ConfigurationQuery
3. The default Configuration

These rules may result in more than one "best" matching Configuration. For example, two Configurations, each having exactly three parameters, might each have two parameters matching ConfigurationQuery parameters. In this case, there is no unique "best" match.

See Also:

`Configuration`, `Application`, `Client`

Method Summary

int	`matches(Configuration config)` Determine whether (and in how many parameters) this ConfigurationQuery matches the given Configuration.
java.lang.String	`setParam(java.lang.String name, java.lang.String value)` Specify a parameter of the configuration query.

Method Detail

Appendix A

setParam

```
public java.lang.String setParam(java.lang.String name,
                                 java.lang.String value)
```

Specify a parameter of the configuration query.

Parameters:
`name` - Parameter name
`value` - Parameter value
Returns:
the previous value of the parameter (or null if no previous value existed)

---

matches

```
public int matches(Configuration config)
```

Determine whether (and in how many parameters) this ConfigurationQuery matches the given Configuration.

Parameters:
`config` - A Configuration to compare to
Returns:
If the query matches the config, return the number of matching parameters. Otherwise, return -1.

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Class DuplicateConfigurationException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--com.sun.lhs.service.transformregistry.DuplicateConfigurationExcepti
```

public class **DuplicateConfigurationException**
extends java.lang.Exception

Exception thrown by `Configuration.setParam()` and `Configuration.removeParam()` when the change is made to an unnamed Configuration and would result in the Configuration becoming a duplicate of another Configuration of the same client.

For example, if part of the registry is as follows:

```
<CLIENT name="my client" description="">
    <CONFIGURATION>
        <PARAM name="foo" value="1"/>
    </CONFIGURATION>
    <CONFIGURATION>
        <PARAM name="foo" value="2"/>
    </CONFIGURATION>
    <CONFIGURATION>
        <PARAM name="foo" value="1"/>
        <PARAM name="bar" value="2"/>
    </CONFIGURATION>
</CLIENT name="my client" description="">
```

then either of the following would cause a DuplicateConfigurationException:

```
configuration.setParam("foo", "1"); // called on the second configuration
configuration.removeParam("bar"); // called on the third configuration
```

because either change would make the modified configuraion equivalent to the first configuration.

See Also:
Serialized Form

Appendix A

Constructor Summary

DuplicateConfigurationException()
DuplicateConfigurationException(java.lang.String msg)

Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

DuplicateConfigurationException

```
public DuplicateConfigurationException()
```

DuplicateConfigurationException

```
public DuplicateConfigurationException(java.lang.String msg)
```

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Class ElementAttachedException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--com.sun.lhs.service.transformregistry.ElementAttachedException
```

public class **ElementAttachedException**
extends java.lang.Exception

Exception thrown by `Application.addClient()` and `Client.addConfiguration()` when the element being added has already been added elsewhere.

For example, the second call would throw an ElementAttachedException:

```
application1.addClient(client); application2.addClient(client);
```

See Also:
`Application`, `Client`, Serialized Form

Constructor Summary

`ElementAttachedException()`
`ElementAttachedException(java.lang.String msg)`

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ElementAttachedException

```
public ElementAttachedException()
```

Appendix A

ElementAttachedException

```
public ElementAttachedException(java.lang.String msg)
```

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Interfaces

Application
Client
Configuration
ConfigurationQuery
TransformationRegistryFactory
TransformationRegistryService

Exceptions

DuplicateConfigurationException
ElementAttachedException
RegistryDefinitionException

Appendix A

Package com.sun.lhs.service.transformregistry

The Transformation Registry provides a facility to to partition and register URLs, devices and their configurations and the transformations that they require.

See:
Description

Interface Summary	
Application	An Application is the top-level node of the Transformation Registry hierarchy.
Client	Clients belong to an Application and specify configurations.
Configuration	A Configuration of a Client defines a set of transcodings (XSL transformations) to be used when Clients request data.
ConfigurationQuery	A ConfigurationQuery is used to fill in all the known configuration parameters, and to look for matching Configurations in the Client.
TransformationRegistryFactory	A factory for Transformation Registry elements.
TransformationRegistryService	The TransformationRegistryService stores the XSL transformations necessary for various client configurations and applications.

Exception Summary	
DuplicateConfigurationException	Exception thrown by `Configuration.setParam()` and `Configuration.removeParam()` when the change is made to an unnamed Configuration and would result in the Configuration becoming a duplicate of another Configuration of the same client.
ElementAttachedException	Exception thrown by `Application.addClient()` and `Client.addConfiguration()` when the element being added has already been added elsewhere.
RegistryDefinitionException	Exception thrown by `TransformationRegistryService.publish()` and `TransformationRegistryService.publishApplication()` when the URL does not point to a valid configuration document.

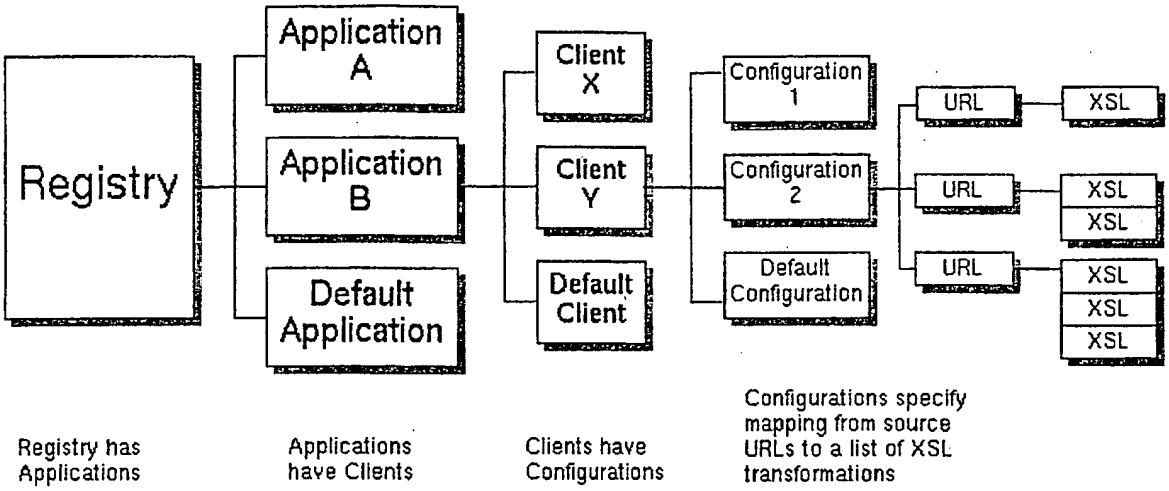
Package com.sun.lhs.service.transformregistry Description

The Transformation Registry provides a facility to to partition and register URLs, devices and their configurations and the transformations that they require. Services that may require transformations can query the registry for the necessary transformations, rather than hard-coding them within applications or services. Transformations can also be performed automatically by the XML proxy for registered clients.

Transformation Registry Data Model

The transformation registry can be represented as a tree:

Appendix A



The registry can be queried in accordance with this data model: each node can provide information about itself (its name and description), and about its children.

The registry can also be altered in accordance with this data model: chilren can be added or removed from each node.

Lastly, the registry (or any single application) may be published by using an XML document to describe the registry (or application) contents.

Transformation Registry API

The Transformation Registry API has three categories of method calls:

- Return information about a node. For example:
  - getName()
  - getDescription()
- Return information about a node's children. For example:
  - getDefault*Child*
  - get*Child*ByName
  - get*Children*
- Add/remove children. For example:
  - add*Child*
  - remove*Child*

Transformation Registry Usage

For an application to use the registry, it must first register itself. It can do so either by:

- Providing an XML document and calling `TransformationRegistryService.publishApplication()`,
- Programmatically creating the tree (using the TransformationRegistryFactory) and calling `TransformationRegistryService.addApplication()`

The application should remove itself from the registry when it stops running.

After it has been registered, an application may query the registry when it receives requests from clients. It does so by getting its Application element from the registry using

Appendix A

```
TransformationRegistryService.getApplicationByName()
```

then choosing a client and a client configuration, and finally specifying the requested URL. This provides the application with a list of XSL transformations to be applied. The application may then use the XSLT service to perform those transformations.

Overview Package Class Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Appendix A

Hierarchy For Package com.sun.lhs.service.transformregistry

Package Hierarchies:
All Packages

Class Hierarchy

- class java.lang.Object
  - class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception
      - class com.sun.lhs.service.transformregistry.**DuplicateConfigurationException**
      - class com.sun.lhs.service.transformregistry.**ElementAttachedException**
      - class com.sun.lhs.service.transformregistry.**RegistryDefinitionException**

Interface Hierarchy

- interface com.sun.lhs.service.transformregistry.**Application**
- interface com.sun.lhs.service.transformregistry.**Client**
- interface com.sun.lhs.service.transformregistry.**Configuration**
- interface com.sun.lhs.service.transformregistry.**ConfigurationQuery**
- interface com.sun.servicespace.**Service**
  - interface com.sun.lhs.service.transformregistry.**TransformationRegistryService**
- interface com.sun.lhs.service.transformregistry.**TransformationRegistryFactory**

Appendix A

com.sun.lhs.service.transformregistry

Class RegistryDefinitionException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--com.sun.lhs.service.transformregistry.RegistryDefinitionException
```

public class **RegistryDefinitionException**
extends java.lang.Exception

Exception thrown by `TransformationRegistryService.publish()` and `TransformationRegistryService.publishApplication()` when the URL does not point to a valid configuration document.

See Also:
`TransformationRegistryService`, Serialized Form

Constructor Summary

`RegistryDefinitionException()`
`RegistryDefinitionException(java.lang.String msg)`
`RegistryDefinitionException(java.util.Vector errors)`

Method Summary

`java.util.Enumeration`	`getErrors()`

Methods inherited from class java.lang.Throwable

`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Appendix A

Constructor Detail

RegistryDefinitionException

```
public RegistryDefinitionException()
```

RegistryDefinitionException

```
public RegistryDefinitionException(java.lang.String msg)
```

RegistryDefinitionException

```
public RegistryDefinitionException(java.util.Vector errors)
```

Method Detail

getErrors

```
public java.util.Enumeration getErrors()
```

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Interface TransformationRegistryFactory

public interface **TransformationRegistryFactory**

A factory for Transformation Registry elements.

The description arguments are optional, meaning that they may be null.

Method Summary

Application	`createApplication(java.lang.String name, java.lang.String description)` Create an application.
Client	`createClient(java.lang.String name, java.lang.String description)` Create a client.
Configuration	`createConfiguration(java.lang.String name, java.lang.String description)` Create a configuration.
ConfigurationQuery	`createConfigurationQuery()` Create a configuration query.
Application	`createDefaultApplication(java.lang.String description)` Create a default application.
Client	`createDefaultClient(java.lang.String description)` Create a default client.
Configuration	`createDefaultConfiguration(java.lang.String description)` Create a default configuration.

Method Detail

createApplication

```
public Application createApplication(java.lang.String name,
                                     java.lang.String description)
```

Create an application.

Parameters:
`name` - The application's name (must be non-null)
`description` - The application's description (optional)
Throws:
java.lang.IllegalArgumentException - if the name is null

Appendix A

createDefaultApplication

```
public Application createDefaultApplication(java.lang.String description)
```

Create a default application.

Parameters:
`description` - The default application's description (optional)

createClient

```
public Client createClient(java.lang.String name,
                           java.lang.String description)
```

Create a client.

Parameters:
`name` - The client's name (must be non-null)
`description` - The client's description (optional)
Throws:
java.lang.IllegalArgumentException - if the name is null

createDefaultClient

```
public Client createDefaultClient(java.lang.String description)
```

Create a default client.

Parameters:
`description` - The default client's description (optional)

createConfiguration

```
public Configuration createConfiguration(java.lang.String name,
                                         java.lang.String description)
```

Create a configuration.

Parameters:
`name` - The configuration's name (optional)
`description` - The client's description (optional)

createDefaultConfiguration

```
public Configuration createDefaultConfiguration(java.lang.String description)
```

Create a default configuration.

Appendix A

Parameters:
`description` - The default client's description (optional)

---

createConfigurationQuery

`public ConfigurationQuery createConfigurationQuery()`

Create a configuration query.

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

com.sun.lhs.service.transformregistry

Interface TransformationRegistryService

public interface **TransformationRegistryService**
extends Service

The TransformationRegistryService stores the XSL transformations necessary for various client configurations and applications. There are three ways to modify the registry:

- Publish an XML document (via the servlet or the `publish()` API) that fully specifies the contents of the registry. A Registry DTD for such documents and an example are available.
- Publish an XML document (via the `publishApplication()` API) that specifies the subtree for a particular application. This is similar to `publish()`, except that only a single application is added or modified. This is particularly useful for applications adding or updating themselves in the registry. An Application DTD for such documents and an example are available.
- Add or delete registry elements (Applications, Clients, etc.) programatically using the API from a service running on the LHS. This is particularly useful to an application, running as a service on the LHS, that would like to add or remove itself from the registry at start/stop time.

The `TransformationRegistryService` runs as a Service in the LHS. It also runs as a servlet on the LHS, to allow an administrator to browse the registry contents, publish a new document and query the registry.

See Also:
`Service`

Appendix A

Method Summary	
boolean	addApplication(Application application) Add an application to the registry.
Application	getApplicationByName(java.lang.String name)
java.util.Enumeration	getApplications()
Application	getDefaultApplication()
java.lang.String	getDomainName() Get the domain name used to resolve hostnames.
TransformationRegistryFactory	getFactory() Get a factory for creating registry elements.
java.lang.String	getLocalhost() Get the machine on which the registry is running.
void	publish(java.net.URL configurationDocument) Publish a document to the TransformationRegistryService that specifies the XSL transformations to be applied for various clients, applications and configurations.
void	publishApplication(java.net.URL configurationDocument) Publish an application.
Application	removeApplication(java.lang.String applicationName) Remove an Application from the registry.

Method Detail

publish

```
public void publish(java.net.URL configurationDocument)
             throws RegistryDefinitionException
```

Publish a document to the TransformationRegistryService that specifies the XSL transformations to be applied for various clients, applications and configurations. If the configuration document is valid, it is used to build the TransformationRegistryService.

Parameters:

configurationDocument - URL of XML document conforming to the TransformationRegistryService DTD

Throws:

RegistryDefinitionException - when the configurationDocument is not valid

---

publishApplication

```
public void publishApplication(java.net.URL configurationDocument)
                    throws RegistryDefinitionException
```

Publish an application. This is similar to publish(), except that only a single application is added

Appendix A

or modified. If the application is already registered, it will be updated from the document.

Parameters:
`configurationDocument` - URL of XML document conforming to the Application DTD
Throws:
RegistryDefinitionException - when the configurationDocument is not valid

---

addApplication

```
public boolean addApplication(Application application)
```

Add an application to the registry. This call fails when there is already an application with the same name in the registry.

Parameters:
`application` - The Application to add to the registry
Returns:
true on success, false on failure

---

removeApplication

```
public Application removeApplication(java.lang.String applicationName)
```

Remove an Application from the registry.

Parameters:
`applicationName` - The name of the application to remove
Returns:
the application that was removed (null if no such application was in the registry)

---

getApplications

```
public java.util.Enumeration getApplications()
```

Returns:
An enumeration of all registered Applications (including default Application, if it exists)

---

getDefaultApplication

```
public Application getDefaultApplication()
```

Returns:
The default Application, if it exists

---

getApplicationByName

```
public Application getApplicationByName(java.lang.String name)
```

Appendix A

Parameters:
name - The name of the Application
Returns:
The Application with the given name

---

getFactory

```
public TransformationRegistryFactory getFactory()
```

Get a factory for creating registry elements.

Returns:
A TransformationRegistryFactory

---

getLocalhost

```
public java.lang.String getLocalhost()
```

Get the machine on which the registry is running. This is a configurable parameter of the Transformation Registry Service and should be configured, using the Bundle Configuration Utility, before the registry is started.

---

getDomainName

```
public java.lang.String getDomainName()
```

Get the domain name used to resolve hostnames. This is a configurable parameter of the Transformation Registry Service and should be configured, using the Bundle Configuration Utility, before the registry is started.

Appendix A

com.sun.lhs.service.clientlookup

Interface ClientLookupService

public interface **ClientLookupService**
extends Service

A companion service to the Transformation Registry for determining the correct Client and Configuration for a User-Agent.

This service is for use by any client of the Transformation Registry, including the Xml Proxy (when enabled).

Method Summary

Configuration	`getUserAgentMapping(Application application, java.lang.String userAgent)` Parses the User-Agent string to determine the Client and Configuration that best matches the User-Agent.

Method Detail

getUserAgentMapping

```
public Configuration getUserAgentMapping(Application application,
                                         java.lang.String userAgent)
                                  throws MultipleMappingException
```

Parses the User-Agent string to determine the Client and Configuration that best matches the User-Agent.

The User-Agent format is `name/version`. The `name` is used to determine the Client (with the same name). The `version` is used to determine the Configuration of the Client.

For non-Mozilla clients (those whose `name` is not "Mozilla"), the version string is used to find a configuration with that name, or a configuration with that version parameter. If neither exists, the default configuration is used.

For Mozilla clients (those whose `name` is "Mozilla", typically browsers), the version string is parsed. If it conforms to either HotJava, Navigator or MSIE format, the version string is parsed for the following parameters and a best-matching configuration is selected (according to the "best match"

rules for Transformation Registry Configurations):

Parameter name	Parameter meaning	Example
`type`	the browser's type	"HotJava"
`version`	the browser's version	"3.0"
`osname`	the operating system name	"Solaris"
`osversion`	the operating system version	"2.x"
`osarch`	the operating system architecture	"sparc"
`lang`	the locale or language	"en" (for English)

Alternatively, if a configuration exists whose version parameter value exactly matches the version string, that configuration will be selected. If no matching configuration exists, the default configuration is used.

Parameters:

`application` - An Application in which to look for the client
`userAgent` - The `User-Agent` header field of an HTTP request

Throws:

MultipleMappingException - when there are multiple Configurations that match the User-Agent string equally well

---

Appendix A

com.sun.lhs.service.clientlookup

Class MultipleMappingException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--com.sun.lhs.service.clientlookup.MultipleMappingException
```

public class **MultipleMappingException**
extends java.lang.Exception

Exception thrown when the ClientLookupService finds multiple configurations matching a User-Agent string

See Also:
Serialized Form

Constructor Summary

MultipleMappingException(java.lang.String userAgent, java.util.Enumeration mappings)

Method Summary

java.util.Enumeration	getMappings()

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Appendix A

MultipleMappingException

```
public MultipleMappingException(java.lang.String userAgent,
                                java.util.Enumeration mappings)
```

Method Detail

getMappings

```
public java.util.Enumeration getMappings()
```

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix A

Claims:

1. A method for providing transformations, comprising:

permitting developers to publish stylesheets in a registry;

maintaining information reflecting a relationship between each of the stylesheets and any devices selected from a set of devices capable of accessing documents rendered in accordance with the stylesheet; and

permitting applications to access the registry.

2. The method of claim 1, wherein permitting developers to publish stylesheets further includes the step of:

providing a set of interfaces to create an application object in the registry, wherein the application object includes application information, a client object, or a configuration object that relates to a stylesheet.

3. The method of claim 2, wherein permitting developers to publish stylesheets further includes the step of permitting developers to add or delete application objects from the registry using the set of interfaces.

4. The method of claim 1, wherein permitting developers to publish stylesheets further includes the steps of:

receiving a static registry, wherein the static registry includes information corresponding to at least one stylesheet;

and creating the registry from the static registry.

5. The method of claim 1, wherein permitting developers to publish stylesheets further includes the step of permitting developers to add or delete stylesheets from the registry using an interface.

6. The method of claim 1, wherein maintaining information further includes the steps of:

receiving a query for a stylesheet in the registry;

locating the stylesheet in the registry; and

providing a response to the query indicating at least one matching stylesheet.

7. The method of claim 6, wherein locating the stylesheet further includes the steps of:

determining which stylesheet to transmit based on parameters associated with the query; and

transmitting at least one stylesheet to an application based on the determination step.

8. The method of claim 6, wherein receiving a query further includes the step of receiving a default query, wherein the default query corresponds to any client.

9. The method of claim 6, wherein receiving a query further includes the step of receiving a specific query, wherein the specific query corresponds to an application, a client, a configuration, and a URL.

10. The method of claim 6, further comprising the step of:

providing a best matching configuration to the query based on a best matching stylesheet.

11. The method of claim 6, wherein receiving a query further comprises the step of:

receiving a query containing a protocol request identifier, wherein the protocol request identifier is not standardized to a canonical form in the registry.

12. The method of claim 1, wherein the stylesheet is an XSL transformation.

13. The method of claim 1, wherein a stylesheet corresponds to at least one application.

14. The method of claim 1, wherein a servlet permits applications to access the registry and permits developers to publish stylesheets in the registry.

15. A method for providing transformations to applications comprising the steps, executed in a data processing system, of:

receiving a request from an application for a transformation including parameters for the transformation;

locating within a registry at least one transformation within the parameters specified by the request; and

providing the at least one transformation to the application.

16. The method of claim 15, wherein receiving a request further comprises:

at the application, parsing information in the request; and

mapping the request to a canonical form based on content associated with the registry.

17. The method of claim 15, wherein the request is a specific request that corresponds to the application, a client, and a configuration.

18. The method of claim 15, wherein the query includes information associated with a client, and wherein the at least one transformation relates to the client.

19. The method of claim 15, further comprising the steps of:

invoking a service to create an XML document based on the transformation; and

transmitting the XML document to a client.

20. A method for providing an XML document to a client, comprising the steps, executed in a data processing system, of:

receiving a request from a client for an XML document;

querying a transformation registry service for an XSL transformation corresponding to the client request;

receiving from the transformation registry service information corresponding to an XSL transformation; and

invoking an XSLT transformation service to apply the received XSL transformation to the XML document.

21. The method of claim 20, further comprising the step of requesting a listing of available XSL transformations from the transformation registry.

22. The method of claim 20, further comprising the step of registering XSL transformations in the transformation registry.

23. The method of claim 20, wherein the request includes information associated with the client.

24. The method of claim 20, further comprising the steps of:

parsing the request based on a protocol request identifier; and

mapping the protocol request identifier to a canonical form to identify a stylesheet in the transformation registry service.

25. A system for providing transformations, comprising:

a memory containing a program that permits developers to publish stylesheets in a registry, that maintains information reflecting a relationship between each of the stylesheets and any devices selected from a set of devices

capable of accessing documents rendered in accordance with the stylesheet, and that permits applications to access the registry; and

a processor configured to run the program.

26. The system of claim 25, wherein the program further provides a set of interfaces to create an application object in the registry, wherein the application object includes application information, a client object, or a configuration object that relates to a stylesheet.

27. The system of claim 26, wherein the program further permits developers to add or delete application objects from the registry using the set of interfaces.

28. The system of claim 27, wherein the program further receives a static registry, and wherein the static registry includes information corresponding to at least one stylesheet and creates the registry from the static registry.

29. The system of claim 27, wherein the program further permits developers to add or delete stylesheets from the registry using an interface.

30. The system of claim 27, wherein the program further receives a query for a stylesheet in the registry, locates the stylesheet in the registry, and provides a response to the query indicating at least one matching stylesheet.

31. The system of claim 30, wherein the program further determines which stylesheet to transmit based on parameters associated with the query, and transmits at least one stylesheet to an application based on the determination.

32. The system of claim 30, wherein the program further receives a default query, wherein the default query corresponds to any client.

33. The system of claim 30, wherein the program further receives a specific query, wherein the specific query corresponds to an application, a client, a configuration, and a URL.

34. The system of claim 30, wherein the program further provides a best matching configuration to the query based on a best matching stylesheet.

35. The system of claim 29, wherein the program further receives a query containing a protocol request identifier, wherein the protocol request identifier is not standardized to a canonical form in the registry.

36. The system of claim 24, wherein the stylesheet is an XSL transformation.

37. The system of claim 24, wherein a stylesheet corresponds to at least one application.

38. The system of claim 24, wherein the program uses a servlet to permit applications to access the registry and permits developers to publish stylesheets in the registry.

39. A system for providing transformations to applications, comprising:

receiving means for receiving a request from an application for a transformation including parameters for the transformation;

locating means for locating within a registry at least one transformation within the parameters specified by the request; and

providing means for providing the at least one transformation to the application.

40. The system of claim 39, wherein the request is a generic request that corresponds to the application.

41. The system of claim 39, wherein the request is a specific request that corresponds to the application, a client, and a configuration.

42. The system of claim 39, wherein the query includes information associated with a client, and wherein the at least one transformation relates to the client.

43. The system of claim 39, further comprising:

invoking means for invoking a service to create an XML document based on the transformation; and

transmitting means for transmitting the XML document to a client.

44. A system for providing an XML document to a client, comprising:

a transformation registry service that receives queries for an XSL transformation corresponding to a client request;

an application that receives requests from a client for an XML document and that receives from the transformation registry service information corresponding to an XSL transformation; and

an XSLT transformation service that applies the received XSL transformation to the XML document.

45. The system of claim 40, wherein the application further requests listings of available XSL transformations from the transformation registry service.

46. The system of claim 44, wherein the application further registers XSL transformations in the transformation registry service.

47. The system of claim 44, wherein the request includes information associated with the client.

48. The system of claim 44, wherein the application further:

parses the request based on information associated with the application; and

maps a protocol request identifier associated with the request to a canonical form to identify a stylesheet in the transformation registry service.

49. A computer readable medium for controlling a data processing system to perform a method for providing transformations executed in a data processing system, the computer readable medium comprising:

a permitting module for permitting developers to publish stylesheets in a registry;

a maintaining module for maintaining information reflecting a relationship between each of the stylesheets and any devices selected from a set of devices capable of accessing documents rendered in accordance with the stylesheet; and

a permitting module for permitting applications to access the registry.

50. The computer readable medium of claim 49, wherein the permitting module for permitting developers to publish stylesheets further includes:

a providing module for providing a set of interfaces to create an application object in the registry, wherein the application object includes application information, a client object, or a configuration object that relates to a stylesheet.

51. The computer readable medium of claim 50, wherein the permitting module for permitting developers to publish stylesheets further includes a permitting module for permitting developers to add or delete application objects from the registry using the set of interfaces.

52. The computer readable medium of claim 50, wherein the permitting module for permitting developers to publish stylesheets further includes:

a receiving module for receiving a static registry, wherein the static registry includes information corresponding to at least one stylesheet; and

a creating module for creating the registry from the static registry.

53. The computer readable medium of claim 50, wherein the permitting module for permitting developers to publish stylesheets further includes a permitting module for permitting developers to add or delete stylesheets from the registry using an interface.

54. The computer readable medium of claim 50, wherein the maintaining module further includes:

a receiving module for receiving a query for a stylesheet in the registry;

a locating module for locating the stylesheet in the registry; and

a providing module for providing a response to the query indicating at least one matching stylesheet.

55. The computer readable medium of claim 54, wherein the locating module further includes:

a determining module for determining which stylesheet to transmit based on parameters associated with the query; and

a transmitting module for transmitting at least one stylesheet to an application based on the determination step.

56. The computer readable medium of claim 54, wherein the receiving module further receives a default query, wherein the default query corresponds to any client.

57. The computer readable medium of claim 54, wherein the receiving module further receives a specific query, wherein the specific query corresponds to an application, a client, a configuration, and a URL.

58. The computer readable medium of claim 49, wherein the stylesheet is an XSL transformation.

59. The computer readable medium of claim 49, wherein a stylesheet corresponds to at least one application.

60. The computer readable medium of claim 49, wherein a servlet permits applications to access the registry and permits developers to publish stylesheets in the registry.

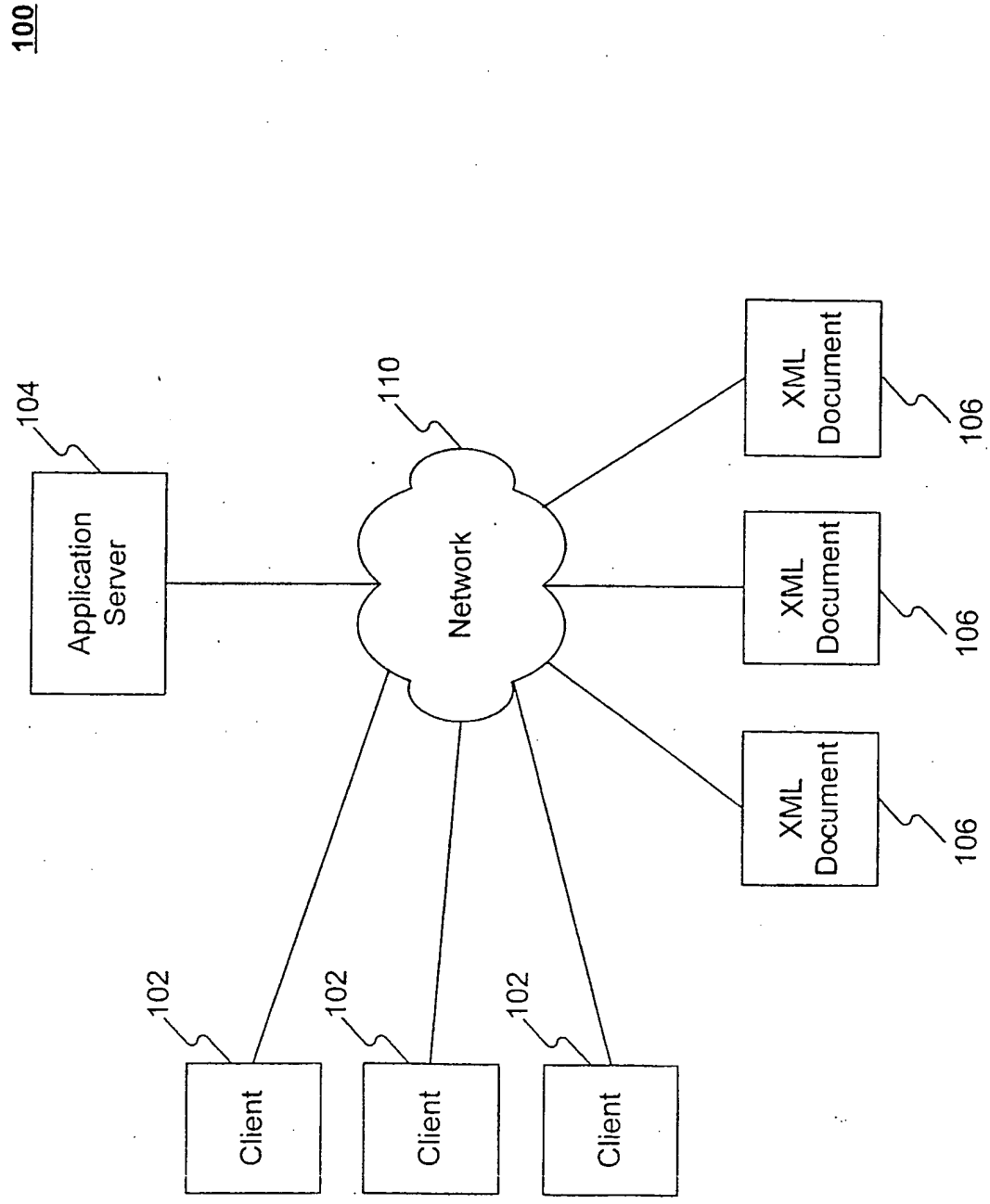

100
104
Application Server
110
Network
XML Document
106
XML Document
106
XML Document
106
102
Client
102
Client
102
Client

FIG. 1

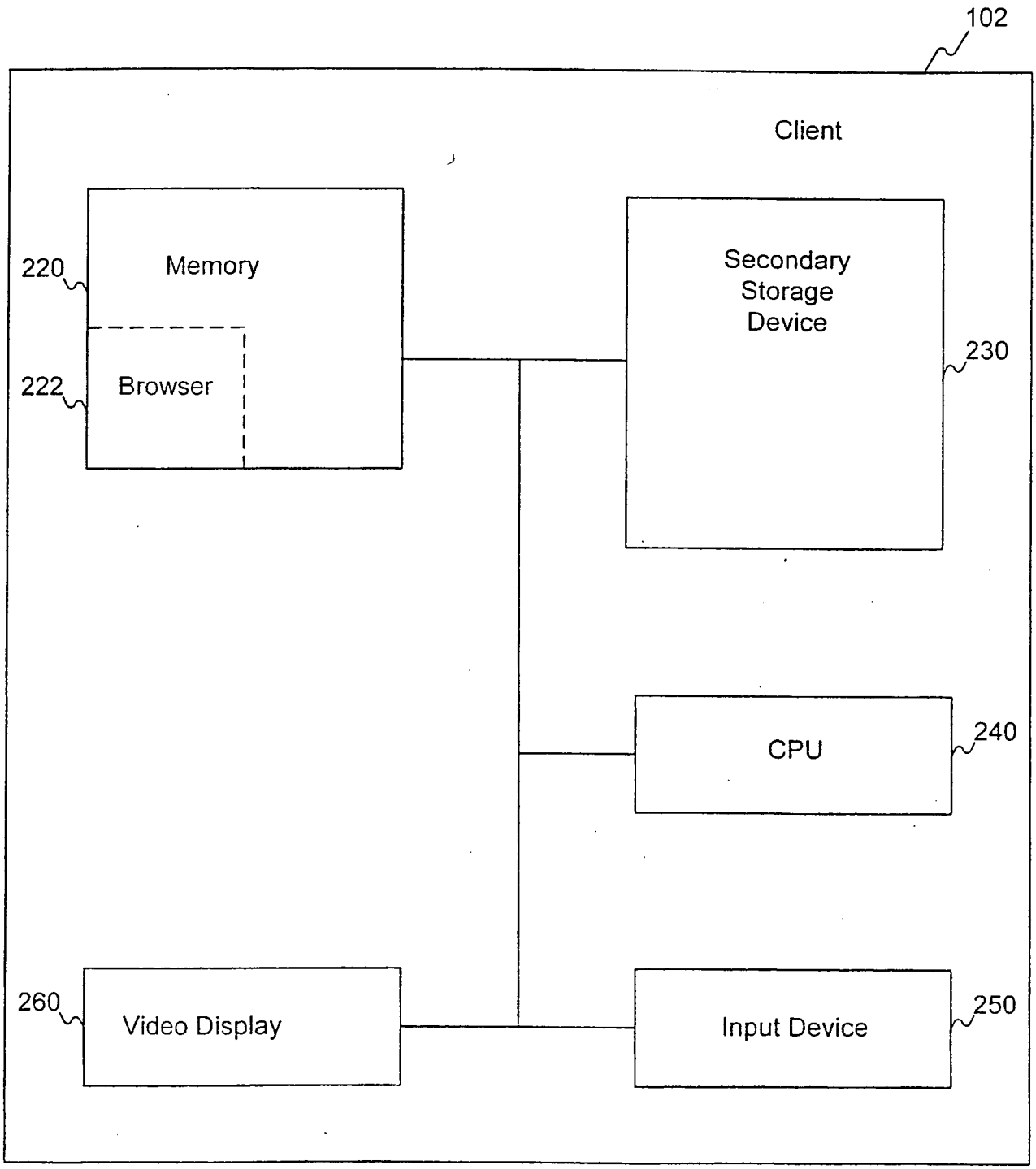

102
Client
220
Memory
222
Browser
Secondary Storage Device
230
CPU
240
260
Video Display
Input Device
250

FIG. 2

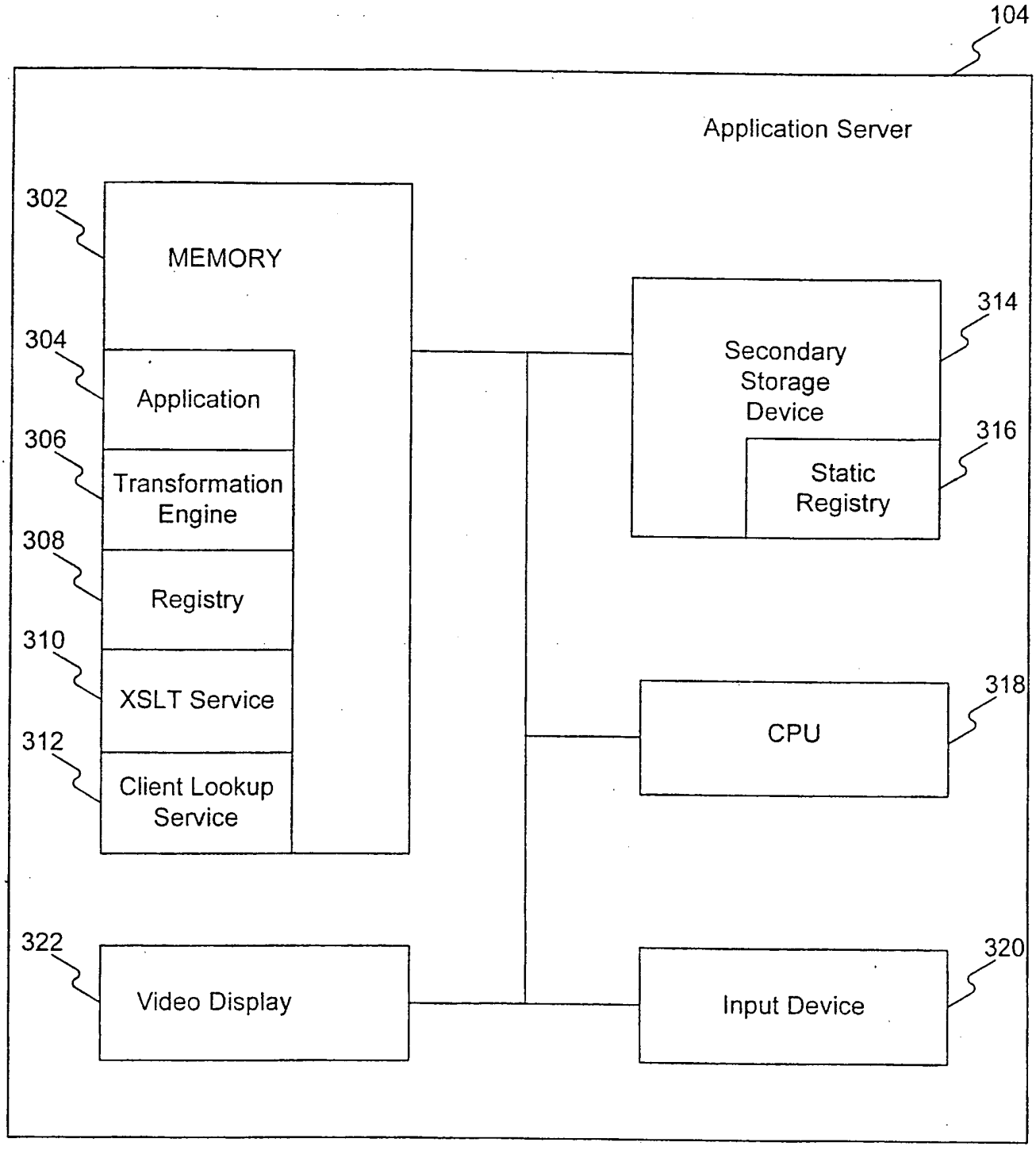

104
Application Server
302
MEMORY
304
Application
306
Transformation Engine
308
Registry
310
XSLT Service
312
Client Lookup Service
314
Secondary Storage Device
316
Static Registry
318
CPU
320
Input Device
322
Video Display

FIG. 3

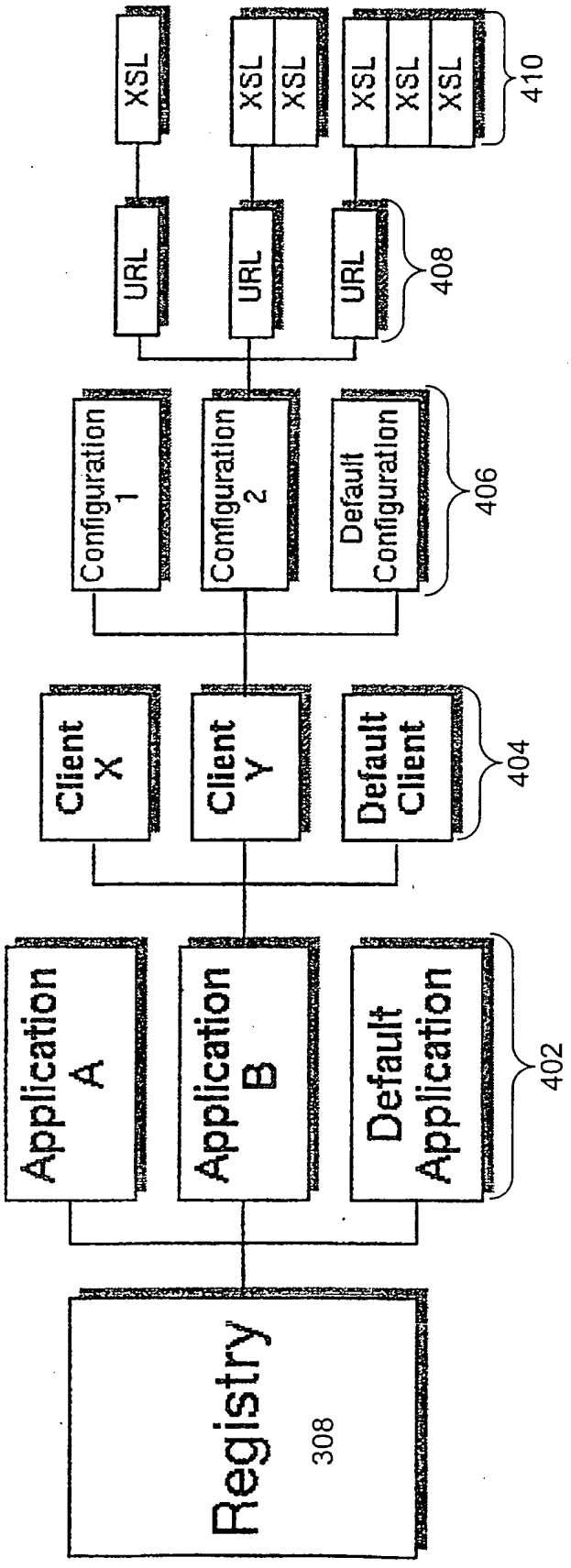

Registry
308
Application A
Application B
Default Application
402
Client X
Client Y
Default Client
404
Configuration 1
Configuration 2
Default Configuration
406
URL
URL
URL
408
XSL
XSL
XSL
XSL
XSL
XSL
410

FIG. 4A

316

```
402 <APPLICATION name="WeatherService" description="Weather and Forecasting Application">
404  <CLIENT name="Mozilla" description="any browser">
406   <CONFIGURATION>
        <TRANSCODING>
408      <SOURCE url="*" />
410      <TRANSFORM
           url="http://localhost:8080/WeatherServiceInfo/examples/weather/weatherservice/resources/mozilla.xsl" />
       </TRANSCODING>
     </CONFIGURATION>
406  <CONFIGURATION>
       <PARAM name="version" value="1.1" />
       <PARAM name="osname" value="Windows CE" />
       <TRANSCODING>
         <SOURCE url="*" />
410      <TRANSFORM
           url="http://localhost:8080/WeatherServiceInfo/examples/weather/weatherservice/resources/wince.xsl" />
       </TRANSCODING>
     </CONFIGURATION>
   </CLIENT>
404  <CLIENT name="sprinkler">
406   <CONFIGURATION name="*">
        <TRANSCODING>
408      <SOURCE url="*" />
410      <TRANSFORM
           url="http://localhost:8080/WeatherServiceInfo/examples/weather/weatherservice/resources/sprinkler.xsl" />
       </TRANSCODING>
     </CONFIGURATION>
404  </CLIENT>
406  <CLIENT name="thermometer">
       <CONFIGURATION name="*">
        <TRANSCODING>
408      <SOURCE url="*" />
410      <TRANSFORM
           url="http://localhost:8080/WeatherServiceInfo/examples/weather/weatherservice/resources/thermometer.xsl" />
       </TRANSCODING>
     </CONFIGURATION>
   </CLIENT>
404  <CLIENT name="weather_vane">
406   <CONFIGURATION name="*">
        <TRANSCODING>
408      <SOURCE url="*" />
410      <TRANSFORM
           url="http://localhost:8080/WeatherServiceInfo/examples/weather/weatherservice/resources/weather_vane.xsl" />
       </TRANSCODING>
     </CONFIGURATION>
   </CLIENT>
404  <CLIENT name="humidity">
406   <CONFIGURATION name="*">
        <TRANSCODING>
408      <SOURCE url="*" />
410      <TRANSFORM
           url="http://localhost:8080/WeatherServiceInfo/examples/weather/weatherservice resources/humidity.xsl" />
       </TRANSCODING>
     </CONFIGURATION>
   </CLIENT>
 </APPLICATION>
```

FIG. 4B

R:\KAMINERM\85021\0253\figs.vsd

316

```
<REGISTRY>

<APPLICATION name="*" description="generic application (or proxy)">
   <CLIENT name="*" description="generic client, like web browser">
     <CONFIGURATION name="*" description="any configuration">
           <!-- no parameters needed for this configuration -->
           <TRANSCODING>
              <SOURCE url="*"/>
              <TRANSFORM url="http://lhs/pretty-print.xsl"/>
           </TRANSCODING>
     </CONFIGURATION>
   </CLIENT>
</APPLICATION>

</REGISTRY>
```

452

FIG. 4C

```
<!-- DTD, Application Publishing, Transformation Registry for LHS -->

<!ELEMENT APPLICATION (CLIENT*)>
    <!ATTLIST APPLICATION name        CDATA #REQUIRED>
    <!ATTLIST APPLICATION description CDATA #IMPLIED>

<!ELEMENT CLIENT (CONFIGURATION*)>
    <!ATTLIST CLIENT name        CDATA #REQUIRED>
    <!ATTLIST CLIENT description CDATA #IMPLIED>

<!ELEMENT CONFIGURATION (PARAM*, TRANSCODING*)>
    <!ATTLIST CONFIGURATION name        CDATA #IMPLIED>
    <!ATTLIST CONFIGURATION description CDATA #IMPLIED>

<!ELEMENT PARAM EMPTY>
    <!ATTLIST PARAM name  CDATA #REQUIRED>
    <!ATTLIST PARAM value CDATA #REQUIRED>

<!ELEMENT TRANSCODING (SOURCE+, TRANSFORM*)>

<!ELEMENT SOURCE EMPTY>
    <!ATTLIST SOURCE url CDATA #REQUIRED>

<!ELEMENT TRANSFORM EMPTY>
    <!ATTLIST TRANSFORM url CDATA #REQUIRED>
```

FIG. 4D

```
<!-- DTD, the Transformation Registry for LHS -->

<!ELEMENT REGISTRY (APPLICATION*)>

<!ELEMENT APPLICATION (CLIENT*)>
    <!ATTLIST APPLICATION name        CDATA #REQUIRED>
    <!ATTLIST APPLICATION description CDATA #IMPLIED>

<!ELEMENT CLIENT (CONFIGURATION*)>
    <!ATTLIST CLIENT name        CDATA #REQUIRED>
    <!ATTLIST CLIENT description CDATA #IMPLIED>

<!ELEMENT CONFIGURATION (PARAM*, TRANSCODING*)>
    <!ATTLIST CONFIGURATION name        CDATA #IMPLIED>
    <!ATTLIST CONFIGURATION description CDATA #IMPLIED>

<!ELEMENT PARAM EMPTY>
    <!ATTLIST PARAM name  CDATA #REQUIRED>
    <!ATTLIST PARAM value CDATA #REQUIRED>

<!ELEMENT TRANSCODING (SOURCE+, TRANSFORM*)>

<!ELEMENT SOURCE EMPTY>
    <!ATTLIST SOURCE url CDATA #REQUIRED>

<!ELEMENT TRANSFORM EMPTY>
    <!ATTLIST TRANSFORM url CDATA #REQUIRED>
```

FIG. 4E

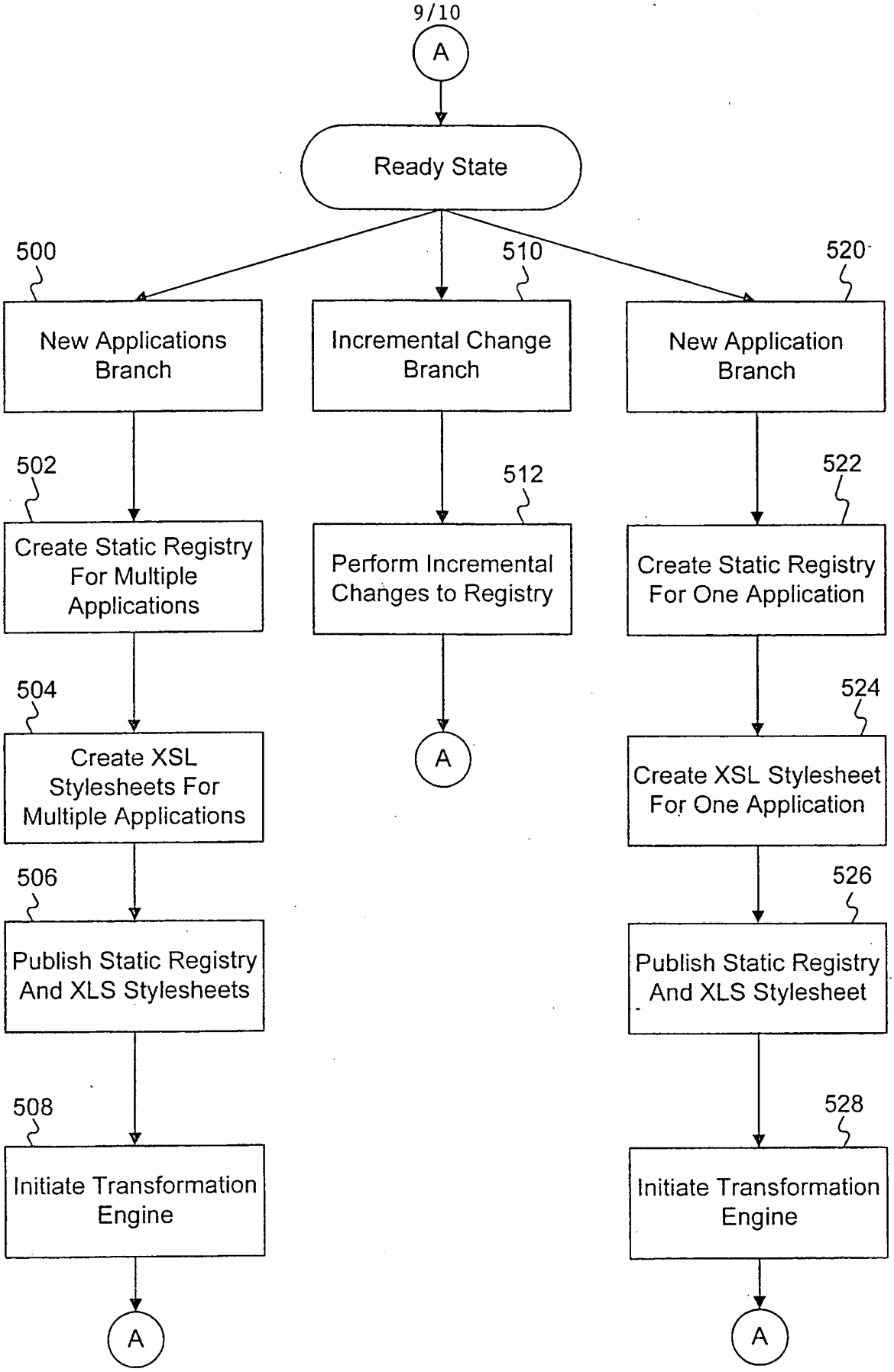

A
Ready State
500
New Applications Branch
510
Incremental Change Branch
520
New Application Branch
502
Create Static Registry For Multiple Applications
512
Perform Incremental Changes to Registry
522
Create Static Registry For One Application
504
Create XSL Stylesheets For Multiple Applications
A
524
Create XSL Stylesheet For One Application
506
Publish Static Registry And XLS Stylesheets
526
Publish Static Registry And XLS Stylesheet
508
Initiate Transformation Engine
528
Initiate Transformation Engine
A
A

FIG. 5

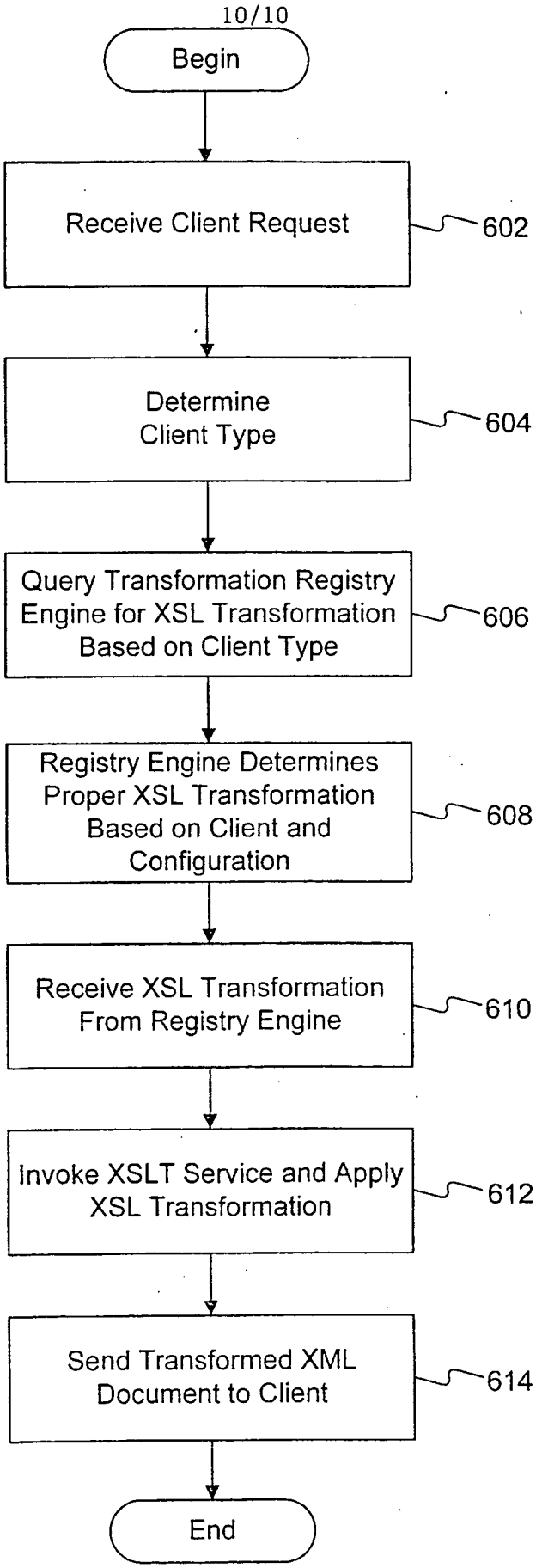

Begin
Receive Client Request
602
Determine Client Type
604
Query Transformation Registry Engine for XSL Transformation Based on Client Type
606
Registry Engine Determines Proper XSL Transformation Based on Client and Configuration
608
Receive XSL Transformation From Registry Engine
610
Invoke XSLT Service and Apply XSL Transformation
612
Send Transformed XML Document to Client
614
End

FIG. 6